Institute of Software Technology
Reliable Software Systems

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Fachstudie

Evaluating Open-source Tool Stacks for Application Performance Diagnostics

Jan Ruthardt
Nico Poier
Thomas Breunig

Course of Study:	Softwaretechnik
Examiner:	Dr.-Ing. André van Hoorn
Supervisor:	Thomas F. Düllmann M.Sc., Teerat Pitakrat, M.Sc.
Commenced:	August 14, 2017
Completed:	February 14, 2018
CR-Classification:	I.7.2

Abstract

Growing system produce an exponential overhead, which can no longer be monitored and observed by a single tool. Otherwise small problems can cause a huge failure. This problem especially occurs at micro-service clusters like Kubernetes, since the number of nodes is bigger than in an monolith systems. To solve this problem, we looked for open-source programs that provide monitoring solutions that cover the whole range of problems including metric monitoring and sending notification to the user. Here the capability of tools to integrate in the environment and to cooperate with other applications was our main focus. We give answers to the question which combination of tools perform the best in a Kubernetes cloud environment that provides micro services. We also speak about usability and maintainability we experienced in the deployment and monitoring of the services.

Contents

Chapter 1

Introduction

Nowadays it is very common in IT to have distributed systems in locations all over the globe[KBKK06]. To provide the best user experience for the System administrators, it is important to monitor these networks by a few people sitting in remote locations. New approaches even try to automate the whole system so that it repairs itself. Important for the user in these systems is the availability and reliability of the system. To tackle these tasks, Application Performance Management (APM) tools were build. They are available in a wide range of costs and qualities. They differ a lot in their architecture, features and style of approach to a problem. This is why we decided to make a comparison of some large Open-Source tool stacks available on the market.

1.1 Goals

The Goal of the study is to print out the benefits and disadvantages of the popular Open-Source tools and stacks for monitoring available on the market. This could be from huge benefit for the new trend of DevOps(Section 1.4) [BWZ15] . With a good monitoring tool, problems in the short intervals of deployment and testing offered by DevOps, could be identified more efficient. In special monitoring systems that log system internals in database, can be used in the debugging phase of an a agile development. The work wants to illustrate the features and technologies of the tools to make it easier for the reader to get an overview of the different software approaches. In particular the tools will be tested in their ability to interact with modern cloud technologies like Docker and Kubernetes. For these technologies, a good control mechanism that is customizable for every user can reduce the time to bug fix and the fear to develop the code. Furthermore, we want to take a look, on the ability of the stacks to integrate in existing environments and support of common tools and interfaces. Moreover, the cross compatibility of the stacks will be tested to the possibility of combining different tools in one environment.

At the end a of the paper the reader should know which monitoring stack or which stacks are optimal for his or her environment.

1.2 Thesis Structure

In the first part, the paper describes the general aspects of monitoring and how we split up tool stacks to take a closer look by the single responsibilities. This part also discusses the characteristics of the environments and their special interfaces. Here also an overview of our test environments is given. After the introduction, the tools will be introduced on their own. We first introduce the tools we actually tested in our study. After we list some tools we can't manage to install. As a conclusion to this work, a general overview of the tool abilities in form of tables and an answer to the question of the best monitoring tools is given. This answer includes more than one tool because there a multiple option that fit on different purposes. Benefit here is that we can clearly exclude some tools out of the list in Section 1.3, because they don't provide enough functions or there capability to integrate in a cloud environment is very poor.

Technical Data: In this chapter, all technical aspects of the test environments and tools are discussed. Moreover, it provides our separation of the different APM activities.

Tools: All tested tool stacks are listed and grouped by companies that developed and support them. If the stack consists of single tools that are usable in stand alone there is an extra section given that describes them and there interfaces. On the end of this chapter is a short list of tool we tested and were not able to deploy on a cluster.

Evaluation: The tested tools are compared in tables after there responsibility(Collector,Database,Visualization,Alerting). Here these tables we compare the statements of the provider with our experience of the tools.

Conclusion: Our experience with the tools and an answer to the question which stack is the best to use in cloud areas like Kubernetes.

1.3 Requirements

In the process of searching for tool we set some minimum requirements that the tools have to fulfill, because over wise the amount of candidate for the study would be to big. The most important requirement was, that the tools are open-source. Moreover the tools had to be able monitor distributed systems and visualize the data. By the term

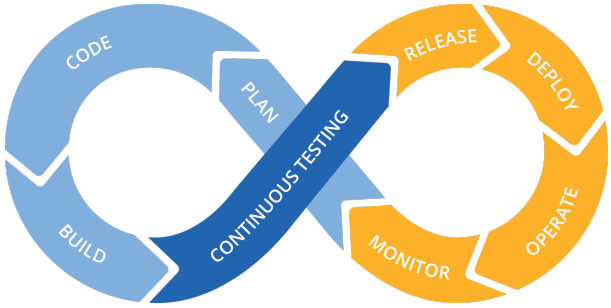

Figure 1.1: An example DevOps cycle [Tri18]

distributed systems we mean a micro service architecture. In case that many of the tools had matched the requirements we defined so nice to have features. We preferred tools that were able to monitor web service over HTTP. Also tools that had an grouping function for alerts or can send the alerts over several interfaces. At the end we had look at the following tools that all matched the minimum requirements of our study: TICK [Infb], ELK Stack [Elab], Prometheus [Clo], Grafana [Graa], Zabbix [Alea], Icinga [Thea], Cacti [Cac], OpenNMS [Opeb], Munin [Mun], Hawkular [haw], OpenTSDB [Opec], Zipkin [Opea]. Of this list we selected the first six tools to take a look on in our study.

1.4 DevOps

DevOps is a scheme for process improvement in the field of Software development and system administration[BWZ15]. It tries to improve the cooperation of both fields by shared process and tools. By a agile style the process is visualized as a cycle Figure 1.1 . Our study trys to improve the monitoring aspect of the cycle.

Chapter 2

Technical Data

In section 2.1 to section 2.5 we talk about how we grouped the responsibility of the tools into categories. In section 2.6 we describe the environment we tested the tools in. We also describe the technologies, which we used to build our testing environment

2.1 General Stack

To explain a Stack in general, distinguish the difference between monitoring and logging. First, monitoring is running in the background and constantly collects the system data, these data is collected by specific metrics. However logging is only triggered by a definite event or exception. With this knowledge it is a easier understanding how to establish a monitoring stack. Therefore 4 different types of tools are needed [HHMO17]. First of all one for collect data from the system by specific metrics, the collector. The second type of tool is to store, maintain and querying them, the database. The third type of tool is to visualize the data, the visualization tool. Sometimes an alerting tool is also needed, but this is often integrated in the visualization tool. Logging stacks have similar architecture, but in this case the Collector does not send metric data to the database. The log files of the whole system are collected in one place. These two types of APM can be done simultaneously.

2.2 Collector

To get data in a centralized spot a tool is needed to collect the data were it is generated and transport it to the server or provide an interface for the server to collect the data. We call tools for this purpose collectors. There are collectors for every monitoring purpose.

It is very common that a collector provides a general interface like XML or JSON files or can be adapted to variable databases to get a wide spectrum of cases to be applied. The monitored metrics dependent on the environment and the collector also has to use other tools that the system provides to get this special type of metrics. In general the data collected can be split up in system data and application data. All system data are all physical values like CPU load, RAM and hard disc drive usage. These will be provided by cAdvisor (2.2.1) in the case of Kubernetes. In the case of monitoring Kubernetes the number of jobs/pods or the number of connection per second/minute will be monitored. These and other data will be provided by the Api-server(2.2.2) of Kubernetes.

2.2.1 cAdvisor

Container Advisor (cAdvisor) is a tool for collecting, processing and exporting data of containers. It is originally designed for Docker but can be applied to any other container. All information about the container is accessible over a REST api that returns a JSON file that contains all data. A copy of cAdvisor is deployed within any Kubernetes Pod, so every APM tool can get the system metrics.

2.2.2 Api-Server

Api-Server is a tool that provides a RESTful interface and is a front end for the whole Kubernetes Cluster. Over the Api-Server a user is able to interact with all components of the cluster.

2.3 Database

The databases for APM are usually time-series based (2.3.1). As every other database it is used to make data persistent and perform request over multiple entities to get new informations about critical values and value changes over a time period. Databases can offer two types of data providing methods. Most of the time the database provides a well defined interface which normally provides an authentication method to insert data into the database. Any snapshot then gets a time stamp. The other method is that the database performs a get operation onto an interface provided by the collector. This type of data collection is better for static systems.

2.3.1 Time-Series-Database

Time-Series-Database is a special kind of database developed for saving time related data. These data consist of arrays which are indexed by a time stamp. By the term Time-Series time ranges could also be used as primary key. These types of Databases are capable of creating, enumerating, updating, deleting and analyzing time-series-data. Often they also allow you to merge multiple time-series together and make one data set out of them. Like each other database, time-series-databases can also filter the data. They are also capable of ordering the dataset by values like time.

2.4 Visualization

The Visualization tools are used to display the data stored in the databases in a comfortable and organized way. This is realized by plain text or graphs. Graphs have the big advantage to be able to display the value changes over time and can very easily illustrate spikes in the data sets. Furthermore graphs can present the data in more than one way, which makes it easier for people to detect abnormal data spikes. Usually all this information can be accessed via a web interface as this also gives an option for identification. This is especially useful when the data are of high importance. Often these tools also implement easy to use interfaces for alerting tools, to set conditions for specific alerts, which can save a lot of time.

2.4.1 Graphs

As previously mentioned, the data we collected from the cluster must be readout of the database and displayed in a readable fashion. Thus most visualization tools use graphs to display the collected data. Using graphs not only makes the data easy to read, but it also adds the option to scale the data to our needs and preferences. This can be very useful when looking for trends in a bigger time range. It also gives the option of color coding the data, which can be useful to either see dangerous values more quickly, or simply render multiple data streams in one graph to compare them or to see them in comparison to the whole system.

2.4.2 Permission Management

Most visualization tools have a web interface in which all the data are displayed. To make sure only authorized people can view the data, these tools usually implement a

few permission management methods. These can be ranging from simple login dialogs to viewing permissions of specific data streams only to authorized people. Some tools allow complete customization of the permission settings, while others offer a set of permission templates. The most popular method of authorization seems to be LDAP, as this can be used for simple and complex permission schemes alike.

LDAP

Written-out Lightweight Directory Access Protocol is a Network-protocol on a client-server basis. LDAP describes the communication between the client and the LDAP Directory. The data-structure of LDAP is the so-called Directory Information Tree which is organized by one suffix(root) and nodes.

2.5 Alerting

To inform the developer about the system, a tool is needed which is able to send warnings about predefined system states. The alerting tool gets one or more error codes from the controller that is normally implemented into the database or visualization tool. Some tools combine these codes. The alerting tool then sends a warning or error message to all people involved. Most of the tools can send over multiple platforms. The most common are: E-Mail, SMS, Telegram and Slack. Many tools offer many of the mentioned interfaces and provide an api to integrate other alerting types. Often the developers don't want to get just one alert with one message, so some of the alerting tool have the possibility to group alerts. With this feature it is possible to group cascading events that are triggered by one failure. In some cases tools also supply an option to divide all alerts into critical alerts and warnings which can help the user to select the important messages.

2.6 Monitoring environment

As many monitoring tools are flexible to the environment we chose a specific setting for the tools to monitor cloud skills (Kubernetes). [Voh16]

2.6.1 Docker

Docker is an Open-Source software that virtualize Operation Systems (OS) with the concept of containers. That means that the application and the operating system is built in to one file and can be deployed out of that file with the before defined settings. This file is called an image. Docker also provides a collection of pre-built images on their page [Doc]. The software is as Kubernetes written in Go and under the Apache License.

2.6.2 Kubernetes

Kubernetes is an Open-Source system that provides a platform for container deployment and management. Its strength is in the field of scaling and maintaining the deployed containers. Kubernetes can run on different hosts at the same time. The Kubernetes master connects all hosts together to form one cluster out of them. The master also works as an interface for other systems to connect to the cluster. From the outside of the cluster it is not visible on which node the application is running. This is realized by a load balancer which is also running on the Kubernetes master. Every container in the cluster also runs a copy of cAdvisor (2.2.1) which is very important for APM, because it collects data from the system like CPU and RAM and provides them over an interface.

2.6.3 GO

Go is a programming language that is as Kubernetes designed, implemented and updated by the Google Inc.. The basic idea about the language is to simplify the syntax compared to c and c++ by preserving their network capability and extend them in the field of cloud technologies. The language itself is capable of all modern concepts such as object orientation and typification and parallelization. It was developed as an Open-Source Software and first released in 2009.

2.6.4 Hardware

We were Testing all the tools in the study on a 3 nodes Kubernetes cluster. Any cluster node has a 4 Core Intel CPU with 2.3 GHz and 4 Megabytes of Cache . The nodes also have 8 Gigabytes of Ram each. The virtual machine is KVM which runs on a Fedora Linux.

2.6.5 Sockshop

The sockshop is a demo-application for micro services, which can be used to test monitoring microservices. It takes up quite a lot of resources, considering it is a website with a small collection of scripts behind it. This makes the sockshop a very good demo-application, as it produces enough data to test monitoring applications, or simply show the ability of a cluster to handle such tasks. Sockshop features a slim but useful HTTP-based API which allows the system owner to retrieve or post data via its own scripts or micro services to complement the application or monitor data values from the inside.

Architecture

Sockshop is written in multiple languages. The Front-end is written in NodeJS to give the user a handy interface. A MySQL database is used to store all the items in the shop, which is especially useful when you have complex data sets for your products. To receive the data from the MySQL database an interface written in GO is used. GO is also used for the retrieval of the users, which are stored in a Mongo database, as here no complex data structures with a lot of connections are needed. The cart of the shop uses Java to store its information inside a Mongo Database. The order process is done via a Java / .NET Core pipeline, which stores the orders inside a MongoDB. The stored orders are then processed by another Java pipeline to determine which orders can be shipped.

Chapter 3

Tools

In this chapter we talk about all tools we decided to test in our study because they fitted best the requirements of Section 1.3. The first part of this chapter handles the tools we managed to install. We used a template to ease the comparison of our experience with the specific tools. The second part contains a list with tools that are not capable of running in Kubernetes. A quick explanation of what went wrong in the installation process is given.

3.1 Searchlight (Icinga)

As the Section 3.7.2 Icinga described, we were not able to find a pure installation of Icinga for a cluster so we tried to install Searchlight as a backup plan. Searchlight is a tool from the AppsCode Inc. [Appa], which is a company located in San Leandro, California. Searchlight is as many other monitoring tools written in the programming language Go.
When first tried to install Searchlight over the yaml File we received the error from the Kubernetes cluster. There it says the tools is not able to bind the Port 8443. We could not figure out which application is using the port but as we tried to install the application on a fresh cluster it turned out working fine.

3.1.1 Appearance

Icinga/Searchlight comes in a discreet blue/white coloring. On the dashboard which represents the homepage of the application an overview of the implemented alerts is shown. The alerts are colored with green/yellow/red for OK/warning/error, so it is easy to recognize appearing problems.

Other functions like history and configurations are located at the sidebar. Information about the monitored host and services are located in an extra row, presented on the right.

3.1.2 Performance

In our VM deployment the application has allocated 160MB of RAM under maximum load. The application it self is running very smooth even on low power systems. As we deployed some alerts, we noticed, that it took about 30 seconds to validate the alert and get a first status from the system. Thirty seconds after deployment, the system shows no reaction. But after this time everything worked fine.

3.1.3 Interoperability

On the Github page [Appb] of the program the developer claims to be able to send notification over email, SMS and chat. In the guide there is no explanation what is meant by the term chat but the tool is able to send notifications to Slack [Sla] and Hipchat [Atl]. Notification over email can be send over SMTP without any third party software. To send SMS, a service like Twilio [Twi] is needed. On the page there is no hint that the tool is capable to interact over an interface with other notification than the given. Furthermore the developer gives no interfaces in at all for communication with other applications.

3.1.4 Conclusion

Searchlight is a light weighted implementation of the famous monitoring tool Icinga. It runs well and has all basic features. As we took a deeper look into the software we were able to uncover its weaknesses. There are no advanced possibilities to connect Searchlight with other monitoring tools. Also there is no graphical user interface for creating alerts, which makes it very difficult to set it up. As the tool is mainly developed and updated by only two people, the support of new versions from Icinga for Kubernetes is limited. The fact that the company Appcode Inc. has no direct correlation with Icinga is another counter-argument for the tool. In a nutshell the tool is good for small Kubernetes clusters that want to monitor only a few basic metrics and have low available resources.

3.2 Prometheus

The tool Prometheus is an open source project and is hosted by the Cloud Native Computing Foundation. It is a single tool that comes with the features of a whole monitoring stack (without alerting).
Prometheus own collector uses client libraries, supported for different languages including GO, Java, Scala, Python, Ruby and more third-party libraries for various other languages. The collector is able to pull data from the server or cluster and can be accessed via HTTP request. The project was originally built at Soundcloud [Sou] and is nowadays used by big companies like Jodel [Theb], Docker and Core OS [Pro]. Prometheus also has an alert manager, which is an extra tool to send message including the cluster state via email, HipChat, PagerDuty, Pushover, Slack, OpsGenie and VictorOps. These alerts have to be set up by a configuration file. Because the project is community driven, there is no official repository for Prometheus on Kubernetes, but with a little research we managed to find a repository [kay] that provided different deployments for cloud environments.

3.2.1 Installation

We managed to install Prometheus in a way that we were able to get metrics from the Kubernetes Api-server and can expose the metrics over a RESTful interface. Over this interface it is possible to connect Prometheus to an application outside or inside the cluster like Grafana, which is recommended for Prometheus by the developers. We were not able to install the Prometheus own alert manager on the cluster. Also we were not able to get metrics from the cluster-nodes because our Prometheus version is not capable of requesting over HTTPS (Hypertext Transfer Protocol Secure), which has to be used for the nodes.

3.2.2 Appearance

Prometheus provides a web user interface to present the collected data. It is designed in a very light weighted black and a white combination with some blue accents and only shows the necessary informations. The interface has no function to save the set of graphs that is selected and no option to authenticate, so that the data can be keeped secret.

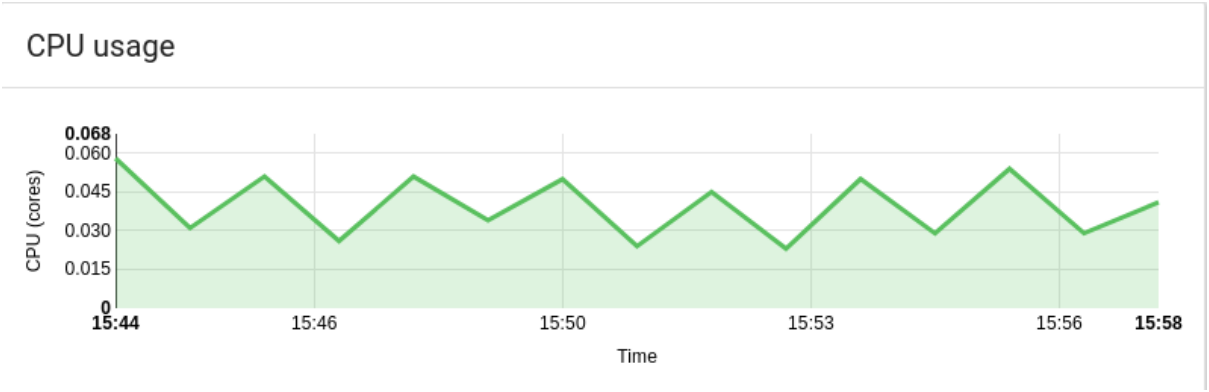

Figure 3.1: CPU usage of Promtheus

3.2.3 Performance

The tool is very quickly booted and shows no performance problems on the interface. A look on the stats shows that Prometheus is not very CPU dependent, it oscillates up and down because the collection data peaks as show in Figure 3.1.Based on the amount of memory usage,the tool is a little bit more demanding and takes up to 2GB of RAM. The high amount of RAM is allocated over a time from about 24h.

3.2.4 Interoperability

It is highly recommended that Prometheus is used with a graphing and an alerting tool, due to the fact that it is just a collector and time series database with a query engine. In our deployment Graphana is used for both tasks, because it is the recommended tool from the Prometheus web page [Clo]. Exploiting Prometheus to any other tool is also possible as Prometheus provides the metrics that are collected in plain text over a RESTful interface.

3.2.5 Conclusion

Prometheus is very good as a collector for large Kubernetes Clusters with hundreds or thousands of nodes. What makes it so good is the with-box monitoring approach, which provides way more metrics than a black-box monitoring. These fact can be used to detect problems in the system earlier and in much more detail so that the downtime can be reduced. On the other side Prometheus is not good for presenting this data and to throw alerts in case of failure.

3.3 Zabbix

[HGS15] The tool Zabbix is an open-source tool and is developed by Zabbix LLC (Limited Liability Company) since 2005 [Alea]. Zabbix provides an all in one solution which includes an collector, a database, a visualization tool and an alert manager.

3.3.1 Installation

There is a repository from monitoringartist [Aleb] that provides diverse yaml files, that map the Zabbix client and server on a cluster structure. The single modules are scalable by the user afterwards in the Cluster. By default 3 copies of the database engine and web user interface are deployed. In the repository are two types of Zabbix deployment. The first we tested uses a load balancer the second one uses the cloud network balancing engine from Google.

3.3.2 Appearance

Zabbix come white a white and blue web interface with some red accents. It implements a tab system with sub tabs to navigate through the interface. The naming of the tabs is not as self explaining as it could be.

3.3.3 Performance

The general performance of Zabbix is very good. The three parts take up to 1GB of RAM form the System and about 0.5 % of the processor load. The minimum requirements of Zabbix are specified with Pentium 2 of 350Mhz and 256MB of Ram [MZ14].

3.3.4 Interoperability

Zabbix provides a different API to connect to other entities. The web interface provides a PHP script that accepts queries and can be configured to use authentications . Grafana has an extra plug-in which is connectable to this script and shows a optimized dashboard with all relevant informations. The tool is also capable of sending alerts over email, SMS, and jabber. Zabbix also gives the ability to create custom scripts that can be execute in case of an alert. For example a command line script can be triggered over SSH.

3.3.5 Conclusion

Zabbix is one of the best optimized and extended open-source solutions available. The installation was by far the simplest of all tested tools. The configuration is complex and the naming of the tabs is not at every position as expected. By default Zabbix scales over all of the available nodes but can be scaled up manually. The best way to use Zabbix is to pare it with a Grafana instance which provides the information from Zabbix tighter and with a better overview.

3.4 ELK Stack

The ELK stack is developed by elastic [Elaa]. The stack is a logging tool which is developed in Java and divided up into three parts. First logstash and beats with their sub tools like Metricbeat or Heartbeat for collecting. Second Elasticsearch which queries the data and makes them persistent and last Kibana the visualization tool of the stack.

3.4.1 Logstash/Beats

Logstash is described as an "server-side data processing pipeline"[Elaa]. These piplines are able to process nearly every data (github webhooks, http push/pull, irc, imap ,etc.). While collecting the data Logstash already transforms it into data structures for better retrieval. Beats is a "single-purpose data shipper"[Elaa] and can be understood as a collector.

3.4.2 Elasticsearch

"Elasticsearch is a distributed, RESTful search and analytics engine capable of solving a growing number of use cases"[Elaa].
It makes the data persistent in JSON (JavaScript Object Notation) with its own Query Domain Specific Language. Theses files are named shards and can be spread over multiple Servers.

3.4.3 Kibana

Kibana is a browser based open-source visualization plugin. It is under the Apache license and is written in JavaScript. It features a web interface with a user login. This gives the owner the power to control who is using which features. User logins can be either added manually or via LDAP. Kibana offers a lot of different graphs including classic line graphs, data tables, area graphs, pie charts and bar graphs.

3.4.4 Installation

ELK Stack is deployable at the Kubernetes cluster. There is a solution on GitHub provided by Kubernetes. Just the deploy the given yaml-files and the tools are running.but the necessary configuration files are missing.

3.4.5 Appearance

Kibana is the only tool in the complete Stack with a user interface. It is accessible over the homepage of the application. The site is starting with a discover page, where all logs are listed in a chronological order. The discover page also allows to filter these logs by key words. Managing tools like dev tools, time, monitoring dashboard, etc can be fownd on the left hand sidebar. The dashboard tab, is for creating an own dashboard with the own preferences.

3.4.6 Performance

The tool ELK stack is booting very fast. As seen in Figure 3.2, the CPU usage is very high at the start, but when all service have finished starting, the docker has a low CPU usage. In contrast the RAM consumption at the beginning is low, but when it starts collecting log data's the amount of used RAM is growing very fast (Figure 3.3). The growth was so big, that our Kubernetes cluster broke after a view days because the server was out of memory.

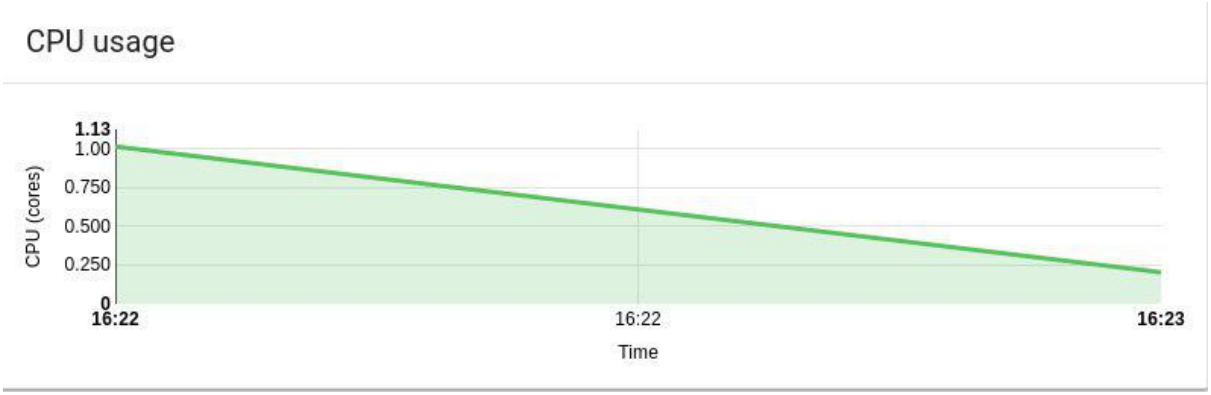

Figure 3.2

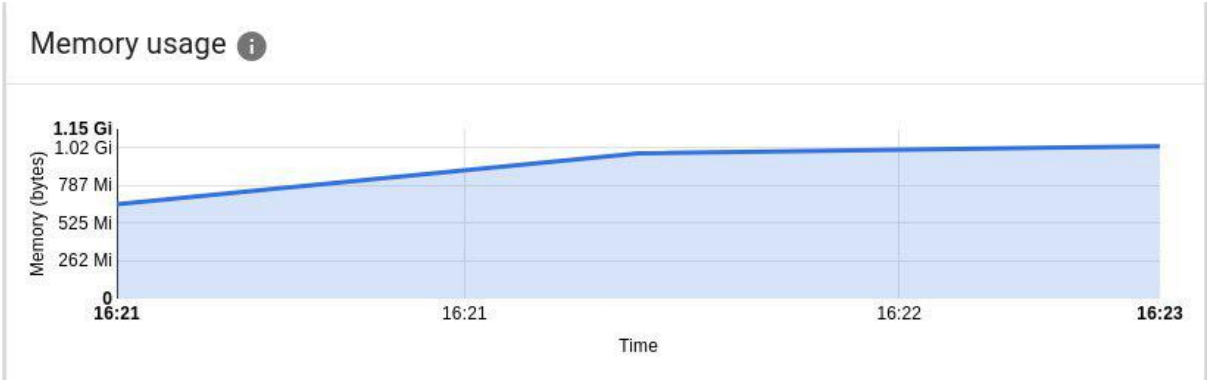

Figure 3.3

3.4.7 Interoperability

Elasticsearch also offers a tool to include alerts. The tool can send any alerts which you can query with Elasticsearch. On the official website, elastics mentioned they can send the notifications on E-Mail, PagerDuty, Slack and HipChat and it also offers an interface with a webhook-output to integrate any third-party-software for messaging. With this interface it might be easy to include sending alerts with SMS like earlier mentioned the program Twilio(3.1).

3.4.8 Conclusion

ELK stack is easy to install but not easy to configure. There is no reference how to properly integrate the tools with each other in Kubernetes. Therefore it is impossible to get the data from Elasticsearch in Kibana, which makes the tool difficult for Kubernetes.

3.5 Grafana

Grafana is a tool to visualize data collected about a system. It features different graphs, including graphs with changes over time, single stats for live feeds, tables to show multiple data from different sources and heat maps to show the distribution of values over a timespan. Grafana also comes with an inbuilt alerting system which allows the user to be notified under specific conditions via E-Mail, Slack, PagerDuty and Kafka. The alerting also has a webhook, allowing the user to send his alerts to a custom endpoint via HTTP. The graphs and alerts are setup in the web interface of Grafana which comes with its own user management. It differentiates between users, administrators and super-administrators. Administrators can be assigned to an organization, allowing different systems to be monitored by one Grafana instance, while still giving the owners of the single system the option to manage their own alerts and graphs. Grafana also features a LDAP authentication to automatically import users and permissions from a user directory.

3.5.1 Installation

The installation of Grafana is very simple, because it only runs on a single Docker container. Grafana is not a collector. Because of this reason, there are many tools which offer an own distribution of Grafana in their repository that is optimized for their data sources.

3.5.2 Appearance

Grafana mainly uses the colors red and orange and comes with a dashboard. Depending on the data source that is selected this changes to a set of graphs which show the information of the data source. Per drag and drop the graphs and text boxes can be freely moved around.

3.5.3 Performance

The performance of Grafana is very good. The tools consumes nearly no CPU and only a few megabytes of RAM as shown in Figure 3.4. The Diagram also shows that the amount of the used resources is constant over time

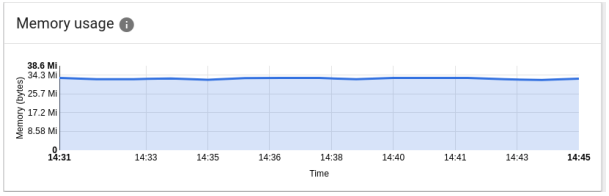

Figure 3.4: Graphane Ressource use

3.5.4 Interoperability

Grafana can interact with nearly every interface to sent alerts because of its web hooks which can send and request to an restful endpoint over a JSON document. Whats special about the alerting engine of Grafana is that pictures can be send within the alerts. The pictures are taken from Grafana to present the alert in a visual way and can be send with the notification or can be uploaded to services like Amazon S3.

3.5.5 Conclusion

Grafana is by far the best and biggest visualization tool on the open-source market. Due to the capability to write custom plug ins nearly every tool stack has an integration which is easy to install and uses the features of Grafana in the intention of the developer.

3.6 TICK

TICK is a Monitoring Stack provided by the InfluxData company and is under MIT License which is a permissive software license.

3.6.1 Telegraf

Telegraf is the data collector of the TICK stack by Influxdata and is written entirely in GO. It is a lightweight data collector as it needs specific plugins for metrics to be collected. These can be chosen by the user as needed for the system. It has official input plug ins for many servers including Apache, Cassandra, Kubernetes, MySQL. This allows the user to send the metrics to almost any endpoint. As with the input plug ins, there is also a broad variety of output plugins including for Influxdatas own InfluxDB, Elasticsearch(3.4.2) from the ELK Stack and Prometheus(3.2). It also supports more primitive outputs like writing to file or TCP/UDP Socket.

3.6.2 InfluxDB

InfluxDB is the time-series database from the TICK stack by Influxdata. It is written in GO and available as a single binary with no external dependencies. This makes the installation, as either a standalone database or as a Kubernetes pod, simple. By default it supports their own collector Telegraf, but other collectors are supported via plug ins. The data can be written with their HTTP/S API using its own SQL-like query language, InfluxQL, which also gives the option of creating your own interfaces. To minimize storage consumption data will be down sampled over time. The recent high precision data is stored in the database only for a limited time and then summarized with other data to keep hard-drive usage reasonable.

3.6.3 Chronograf

Chronograf is the Visualization Tool of the TICK Stack by Influxdata. It comes with a web interface to display all the data collected by Telegraf. Data can be visualized in different graph-types including single and stack line graphs, Step-Plot, Single stat and bar graphs. It comes with a lot of pre-configured dashboards for all the applications which are also officially supported by Telegraf, though it is also possible to create your own dashboards or modify the existing ones. Chronograf also features a UI for Influxdatas own alerting tool Kapacitor, which allows the management of both tools in one interface.

3.6.4 Kapacitor

Kapacitor is the alerting tool of the TICK Stack by Influxdata. Various alerts can already be defined when using Chronograf with Kapacitor including static thresholds, percentage values and deadman switches. It also supports more complex alert conditions via TICKscripts which have to be defined in Kapacitor directly. The alerts it self can then be sent to various endpoints like HipChat, Pagerduty, Pushover, Slack, Telegram and many more. It also supports more simplistic endpoints like file outputs, TCP sockets or HTTP Post. Kapacitor does not feature a web interface as it is meant to be used with Chronograf which already has a UI component for Kapacitor. It can be used without this interface via console inputs.

3.6.5 Installation

The installation of the TICK stack in special Telegraf and Influxdb is very simple. A possible portation of the tool is delivered by the developer of Kubernetes in there Github

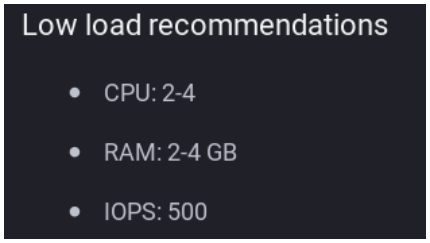

Figure 3.5: Graphane Ressource use

repository Heapster [Hea]. Heapster is used by the stack to collect the monitoring information of the pods in one place.

3.6.6 Appearance

Only Chronograf provides a web interface for the user. The interface is in blue white combination and contains a lot of black. The menu is located on an left sidebar but there are only symbols. On hover the text appears and shows further informations for navigation. It offers a graphical way to inject alerts into Kibana.

3.6.7 Performance

In our cluster the TICK stack performed very well which matches the InfluxData guidelines that recommend a dual core CPU and 2 to 4 GB of RAM (see alos Figure 3.5) as minimum and for system with under five queries per second. Due to the Developer TICK can handle up to over 100 queries but then need 4 to 8 times more resources [Infa].

3.6.8 Interoperability

TICK and in special influxdb can be accessed by many tools. Because TICK stack is very common many tools offer plug-in integrations. In special we tested Grafana because it is recommended by the developer and integrates out of the box. Influx exposes to it not only the physical specs of all nodes, but also the stats collected by Kubernetes to every deployed pod.

3.6.9 Conclusion

The TICK stack is one of the best optimized monitoring solutions for Kubernetes. What makes it special is that information about every running pod instance is collected and can be accessed.

3.7 Failed Tools

In the process of developing and evaluating the APM we discovered a bunch of tools that we were not able to install even that they claimed to be optimized to work on Kubernetes . In this paragraph all the tools we wanted to include in our report but does not work, are mentioned with a quick description of the failure.

3.7.1 Graphite

Graphite is mainly for storing and graphing data and metrics, but brings also tools that are able to collect these metrics from the system. By the developer it self there is no Kubernetes installation provided but there are diverse approaches by third party members to make it runnable on a Cluster. We have tested the Repository from nanit [Grab] to get Graphite running with StatsD [Grac] as a metric collection tool. The Repository does not provide a yaml file by itself to install all the tools. The instruction leads the user to export some variables needed for the installation. After that a deploy command is provided that pulls the docker repository and then installs it with kubectl on the Cluster. As we tried to execute this command an error was thrown, The node replicas were empty, so no further commands are executable. As we were not able to install the tool on multiple Kubernetes Clusters, we installed the tool as on the website advertised on a Ubuntu system directly. With this installation of Graphit a time-series data, a monitoring and alerting tool is included. On the test system the monitoring system gave values that differs from the Linux intern monitoring in values like CPU usage or RAM. As a solution to all this difficulties we decided to not perform further test on the tool.

3.7.2 Icinga

Icinga is as ELK-Stack not a monitoring tool. Its made for logging defined requests. The administrator can decide which system to monitor and in what interval the data is pulled. Icinga it self only provides three system states instead of exact values. The states are

ok,warning and error. The user decides at which point they are triggered. As we tried to install Icinga we found out that there was no direct support from Incinga for Kubernetes. As our Study only describes the actual state without trying to add something we decided not to try an compile Icinga into Kuberntes on our own. Later we found out there is a third party Software called searchlight 3.1 which is provided by Appscode on Github.

Chapter 4

Evaluation

This chapter deals with the comparison of the tools grouped by their responsibility. First, we explain some concepts, we have encountered in our research. These concepts will be covered in Section 4.1. Each table shows the specification the developer gives. They are ordered by the responsibility of tools. The information shown in the tables are from the website of the tool providers. In all tables below we indicated present features with a ✓ and non-present features with a x, if they were advertised by the developer.

4.1 Comparison of concepts

In the scope of this study we identified concepts to group the tools by comparison. Most of them are self explanatory. The ones that are not self explanatory are listed below with a brief introduction.

4.1.1 Push/Pull

There are two basic concepts of the exchange of informations between the collector and the database. Hereby referred as push and pull. Push [HW10] is a concept, were the collector automatically starts sending data to the database as soon the system is started. The data is send continuously. Important is that the collector knows or discovers the data destination, mostly through the use of a discoverer. Pull [HW10] on the other hand means that the database sends requests to all active nodes. These requests are executed in certain intervals. These intervals can be modified and adjusted according to the system requirements. In this concept the database needs a discoverer to find all existing nodes

4.1.2 REST

Representational State Transfer is a programming paradigm for distributed systems. It describes a way, how services in special web services interact with each other. Important concepts of REST are loose coupling, interoperability and scalability. Any data in REST is made persistent in resources. These resources are addressable over an Uniform Resource Identifier.
To make an interface RESTful, it has to provide all HTTP methods like GET, PUT, POST, OPTIONS within a well known data format as XML or JSON.

4.2 Collector

The Table 4.1 compares the different collector tools tested in this paper. The tools are compared by RESTful (Section 4.1.2), Multiple(Multi.) Input Formats, Plug-ins for own data and Push/Pull (Section 4.1.1).

Tool	RESTful	Multi. Input Formats	Plug-ins for own data	Push / Pull
Telegraf	Yes (✓)	Yes	Yes	Pull (✓)
Beats	Yes (✓)	Yes	Yes	Push (✓)
Logstash	Yes (✓)	No	Yes	Push/Pull(✓)
Icinga2	No	No	Yes	Push
Prometheus	No	No	Yes	Pull (✓)/ Push
Zabbix	No	No	No	Both

Table 4.1: Collector-table - All tested collector tools are listed and compared

4.3 Database

Table 4.2 shows the databases. First we checked if the tools have an authentication-based asses to the content. Furthermore we have selected the tools according to time-series-database. In addition to that, we checked, if multiple datasets per time stamp are allowed.

Tool	Database-Authentication	Time-Series-Database	Multi. Data Points
InfluxDB	Yes (✓)	Yes (✓)	Yes (✓)
Elasticsearch	https (✓), credentials (X)	Yes (✓)	Yes (✓)
Icinga2	Yes (✓)	Yes (✓)	Yes (✓)
Prometheus	No	Yes (✓)	Yes (✓)
Zabbix	Yes (✓)	Yes (✓)	Yes (✓)

Table 4.2: Database-table - All tested database tools are listed and compared

4.4 Visualization

Table 4.3 compares the different visualization tools. One aspect is the security of the tool, this is compared by LDAP-Authentication (Section 2.4.2). In general the tools are tested on the base of the graphs that they can display. We tested if they are scalable and it is possible to show multiple graphs in one.

Tool	LDAP-Authentication	Scalable graphs	Overlapping graphs
Chronograf	No	Yes	Yes
Kibana	Yes (✓)	Yes (✓)	Yes (✓)
Grafana	Yes (✓)	Yes (✓)	Yes (✓)
Zabbix	Yes (✓)	Yes (✓)	Yes (✓)

Table 4.3: Visualization-table - All tested visualization tools are listed and compared

4.5 Alerting

The last Table 4.4 and Table 4.5 compares the altering tools. These tools were tested if they can send HTTP requests. Moreover, when many identical alerts arrive, whether they are groupable. Furthermore we tested the most common messaging tools for sending alerts and if the tools offer an interface to integrate any third-party-tool for messaging. Finally, the tools have been tested for triggering events like scripts when an alert is received.

Tool	HTTP requests	Grouping of alerts	E-Mail	Messangers
Kapacitor	Yes	No	Yes	Yes
Elastalert	No	No	Yes (✓)	Yes (✓)
Grafana	Yes (✓)	Yes (✓)	Yes (✓)	Yes (✓)
Zabbix	Yes (✓)	Yes (✓)	Yes (✓)	Yes (✓)
Prometheus	Yes (✓)	Yes (X)	Yes (X)	Yes (X)

Table 4.4: Alerting-table - All tested alerting tools are listed and compared

Tool	Custom alerts	Action on alert
Kapacitor	Yes (✓)	No
Elastalert	Yes (✓)	No
Grafana	Yes (✓)	Yes (✓)
Zabbix	Yes (✓)	Yes (✓)
Prometheus	Yes (✓)	No

Table 4.5: Alerting-table - Table 4.4 cont.

Chapter 5

Conclusion

At the beginning of this paper we asked the question which monitoring tools are the best. In Section 5.1 we want to answer this question. In the Section 5.2 an overview of the often use methods is given. At the end in Section 5.3 an outlook on future work is presented.

5.1 Result

The question on the best tools always depends on the environment that is used. Often a monitoring structure is already existing and new solutions have to be build around it. To answer the question we decided that after the test what we did in our study we can recommend a monitoring stack for a Kubernetes cluster out of the tools we testes. In the test we had the best experience with an combination from Zabbix and Grafana . The installation is very easy and can be done with a single command. There are also different versions of the Zabbix portation. Some of them are more light weighted or have more tools than the others. In terms of functions Zabbix provides nearly everything. To complete this functionality with usability and interoperability we recommend to configure a Grafana instance onto the REST API of Zabbix. This is easily done because Grafana has its own Zabbix plug-in. It provides a very nice dashboard view of the cluster metrics. If there is any reason fur not using a Zabbix installation or if a deeper look into the single nodes and pods is required, we could recommend a combination of Heapster with InfluxDB and Grafana. The installation is slightly more complex because all the tools have to be configured to work together. The huge benefit is that all tools are adopted by the Kubernetes Developers to work great with the cluster. This not only allows a very detailed look on every pod that is deployed, but also ensures a good performance, even in small clusters.

5.2 General Trends

Most of the systems we evaluated or considered evaluating were metric monitoring stacks. In general most of the tools detect not the cause of the failure. Instead the effect is recognized and can be treated manually. Some tools left us with the impression of a trend towards more automation. These tools had functions like automated script triggering over SSH or automatic delivery of logs in the alert message.

Another trend we saw is that alerting managers implement a massive amount of different services for messaging, like slack or telegram, that are used by modern developer teams. Some of the more prominent tools were also capable of REST or SOAP requests to give the option to implement a lot of different interfaces. To keep an overview over a cascading failure, some of the tools can group errors to one alert, thus minimizing the number of alerts.
In Visualization a big trend is the presentation of similar data in one single graph. These method provides a better overview of the system and leads to generalization of the data. Moreover most tools use coloring to highlight warnings or errors. A few tools were also capable of drag and drop graphs, enabling the user to setup a custom dashboard with their graphs.
We also saw some tendency in the database evolution. Databases are no longer just for storing single data points. They now have a greater added value by implementing there own querying language. Using this to purify the collected data for visualization tools. The most established data format is JSON because the files can be fragmented over multiple nodes.

The field of collectors drifts towards multi functional interfaces. This trend is settled, because many systems have to be integrated in already existing monitoring environments. In addition there are two types of collectors. One for collecting logs and processing them and another for the metrics. Both are necessary to monitor a complete system as a whole.

When we installed and tested the tools, all of the above trends were reflected. That is how we identified and worked out the trends.

5.3 Future Work

As a follow up to our work a deeper look into the field of log monitoring tools could provide a more efficient way of failure treatment. An important task is to look where micro service environments like Kubernetes or technology like docker store their logs.

To get the best results from the logs, a good database with the capability of adapting to the quickly changing services is needed. We think a clear mapping of the logs to the services informations is a main quest of the work. As we only looked at solutions that are open-source, there is the possibility to expand our work to the whole software market to take more tools into account. This work could discuss the question of benefits that open source has over paid tools for the new DevOps style of developing.

Appendix

Bibliography

[Alea] Alexei Vladishev, ed. *Zabbix*. URL: https://www.zabbix.com (cit. on pp. 9, 21).

[Aleb] Alexei Vladishev, ed. *Zabbix$_{K}ube$*. URL: https://www.zabbix.com (cit. on p. 21).

[Appa] Appscode, Inc, ed. *Appscode*. San Leandro CA 94578. URL: https://appscode.com/ (cit. on p. 17).

[Appb] Appscode, Inc, ed. *searchlight*. San Leandro CA 94578. URL: https://github.com/appscode/searchlight (cit. on p. 18).

[Atl] Atlassian Corporation plc, ed. *HipChat*. URL: https://de.atlassian.com/software/hipchat (cit. on p. 18).

[BWZ15] L. Bass, I. Weber, L. Zhu. *DevOps: A Software Architect's Perspective*. 1st. Addison-Wesley Professional, 2015. ISBN: 0134049845, 9780134049847 (cit. on pp. 7, 9).

[Cac] Cacti Group, Inc., ed. *Cacti*. URL: http://www.cacti.net (cit. on p. 9).

[Clo] Cloud Native Computing Foundation, ed. *Prometheus*. URL: https://prometheus.io/ (cit. on pp. 9, 20).

[Doc] Docker, Inc., ed. *Docker*. San Francisco, CA 94107. URL: https://www.docker.com/ (cit. on p. 15).

[Elaa] Elastic, ed. *Elasticsearch*. URL: https://www.elastic.co/ (cit. on p. 22).

[Elab] Elasticsearch, ed. *ELK stack*. 90429 Nuremberg. URL: https://www.elastic.co (cit. on p. 9).

[Graa] Grafana Labs, ed. *Grafana*. URL: https://grafana.com (cit. on p. 9).

[Grab] Graphite, ed. *Graphite Kubernetes*. URL: https://github.com/nanit/kubernetes-graphite-cluster (cit. on p. 29).

[Grac] Graphite, ed. *Graphite Kubernetes*. URL: https://github.com/etsy/statsd.git (cit. on p. 29).

[haw] hawkular, ed. *hawkular*. URL: www.hawkular.org (cit. on p. 9).

[Hea] Heapster, ed. *Heapster*. URL: https://github.com/kubernetes/heapster (cit. on p. 28).

[HGS15] J. Hernantes, G. Gallardo, N. Serrano. "IT Infrastructure- Monitoring Tools." In: *the Ieee Computer Society* (2015), pp. 88–93. ISSN: 0740-7459. DOI: 10.1109/MS.2015.96 (cit. on p. 21).

[HHMO17] C. Heger, A. van Hoorn, M. Mann, D. Okanovic. "Application Performance Management: State of the Art and Challenges for the Future." In: *Proceedings of the 8th ACM/SPEC International Conference on Performance Engineering (ICPE '17)*. ACM, 2017. URL: http://eprints.uni-kiel.de/37427/ (cit. on p. 11).

[HW10] H. Huang, L. Wang. "P amp;P: A Combined Push-Pull Model for Resource Monitoring in Cloud Computing Environment." In: *2010 IEEE 3rd International Conference on Cloud Computing*. July 2010, pp. 260–267. DOI: 10.1109/CLOUD.2010.85 (cit. on p. 31).

[Infa] Influx, ed. *Influx Requirements*. URL: https://docs.influxdata.com/influxdb/v0.10/guides/hardware_sizing/ (cit. on p. 28).

[Infb] Influxdata, ed. *Tick*. 90429 Nuremberg. URL: https://www.influxdata.com/ (cit. on p. 9).

[kay] kayrus, ed. *prometheus$_k$ube*. URL: https://github.com/kayrus/prometheus-kubernetes (cit. on p. 19).

[KBKK06] G. Khanna, K. Beaty, G. Kar, A. Kochut. "Application Performance Management in Virtualized Server Environments." In: *2006 IEEE/IFIP Network Operations and Management Symposium NOMS 2006*. Apr. 2006, pp. 373–381. DOI: 10.1109/NOMS.2006.1687567 (cit. on p. 7).

[Mun] Munin Monitoring, ed. *Munin*. URL: http://munin-monitoring.org (cit. on p. 9).

[MZ14] O. Marik, S. Zitta. "Comparative analysis of monitoring system for data networks." In: *2014 International Conference on Multimedia Computing and Systems (ICMCS)* (2014), pp. 563–568. DOI: 10.1109/ICMCS.2014.6911307. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6911307 (cit. on p. 21).

[Opea] Open Zipkin, ed. *Zipkin*. URL: https://zipkin.io/ (cit. on p. 9).

[Opeb] OpenNMS Group, ed. *OpenNMS*. URL: https://opennms.org/en (cit. on p. 9).

[Opec] OpenTSDB, ed. *OpenTSDB*. URL: http://opentsdb.net/ (cit. on p. 9).

[Pro] Prometheus, ed. *Prometheus*. URL: https://prometheus.io/ (cit. on p. 19).

[Sla] Slack Technologies, ed. *Slack*. URL: https://slack.com (cit. on p. 18).

[Sou] Soundcloud limited, ed. *Soundcloud*. URL: https://soundcloud.com (cit. on p. 19).

[Thea] The Icinga Project, ed. *Icinga*. 90429 Nuremberg. URL: https://www.icinga.com/ (cit. on p. 9).

[Theb] The Jodel Venture GmbH, ed. *Jodel*. 10963 Berlin. URL: https://jodel.com/ (cit. on p. 19).

[Tri18] Tricentis GmbH. *Tricentis-What is continuous testing*. https://www.tricentis.com/what-is-continuous-testing/. [Online; accessed 05-February-2018]. 2018 (cit. on p. 9).

[Twi] Twilio, Inc, ed. *Twilio*. URL: https://www.twilio.com/ (cit. on p. 18).

[Voh16] D. Vohra. *Kubernetes Microservices with Docker*. 2016. ISBN: 978-1-4842-1906-5. DOI: 10.1007/978-1-4842-1907-2. URL: http://link.springer.com/10.1007/978-1-4842-1907-2 (cit. on p. 14).

All links were last followed on January 26, 2018.

Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

---

place, date, signature